A LOOP-FREE ALGORITHM BASED ON PREDECESSOR INFORMATION

Shree Murthy and J.J. Garcia-Luna-Aceves
University of California
Santa Cruz, CA 95064
shree, jj@cse.ucsc.edu

Abstract

The loop-free predecessor based routing algorithm (LPRA), is presented. This algorithm eliminates the formation of temporary routing loops without the need for internodal synchronization spanning multiple hops or the specification of complete path information. Like other algorithms, LPRA operates by specifying the second-to-last hop and distance to each destination; in addition, it uses an interneighbor synchronization mechanism. A detailed proof of correctness is presented and its complexity is evaluated. Its performance is compared by simulation with an ideal link-state algorithm and a loop-free algorithm (called DUAL) that is based on internodal coordination spanning multiple hops; both algorithms are representative of the state of the art in distributed routing algorithms. The simulation results show that LPRA is a better alternative than those two algorithms in terms of the number of steps needed to converge after a topological change and is comparable to DUAL in terms of the number of messages and CPU cycles needed to converge after a topological change.

1 Introduction

Many of the routing protocols used in today's networks are based on the distributed Bellman-Ford algorithm (DBF) for shortest-path computation [2]. However, DBF suffers from the *bouncing effect* and the *counting-to-infinity* problems [11]. Recently, distributed shortest-path algorithms [1, 7, 8, 10, 12] that utilize information regarding the length and second-to-last hop (or predecessor) of the shortest path to each destination have been proposed to eliminate the counting-to-infinity problem of DBF. We call these type of algorithms *path-finding algorithms*. Although these algorithms provide a marked improvement over DBF, they do not eliminate the possibility of temporary loops. The loop-free algorithms reported to date rely on mechanisms that require routes either to synchronize along multiple hops [4, 3, 9], or exchange path information that can include all the nodes in the path from source to destination [6].

*This work was supported in part by the Office of Naval Research under Contract No. N-00014-92-J-1807 and by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175

In this paper, we present a path-finding algorithm that is loop-free at every instant. We call this algorithm the *loop-free predecessor-based routing algorithm* (LPRA). According to LPRA, update messages are sent only to neighboring nodes. LPRA eliminates all temporary loops by implementing an interneighbor coordination mechanism with which potential temporary loops are blocked before routers can forward data through them. To block a potential temporary loop, a node sends a query to all its neighbors reporting an infinite distance to a destination, before changing its routing table; the node is free to choose a new successor only when it receives the replies from its neighbors. To reduce the communication overhead incurred with interneighbor coordination, nodes use a *feasibility condition* to limit the number of times when they have to send queries to their neighbors. In contrast to many prior loop-free routing algorithms [3, 4, 9], queries propagate only one hop in LPRA. Furthermore, updates and routing-table entries in LPRA require a single node identifier as path information, rather than a variable number of node identifiers as in prior algorithms [6].

Section 2 presents the network model assumed in LPRA. Section 3 provides a description of the algorithm and an example illustrating key aspects of its operation. Section 4 provides a detailed proof of LPRA's correctness. Section 5 addresses the performance of LPRA. Finally, Section 6 presents our conclusions.

2 Network Model

A computer network is modeled as an undirected graph represented as $G(V, E)$, where V is the set of nodes and E is the set of links connecting the nodes. Each node represents a router and is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. A functional bidirectional link connecting the nodes i and j is represented as (i, j) and is assigned a positive weight in each direction. The link is assumed to exist in both the directions at the same time. All the messages received (transmitted) by a node are put in the input (output) queue on a first-come-first-serve basis and are processed in that order. An underlying protocol assures that: (a) every node knows who its neighbors are; this implies that a node within a finite time detects the existence of a new neighbor or the loss of connectivity with a neighbor. (b) all packets transmitted over an operational link are received correctly and in the proper sequence within a finite time; and (c) all

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2006		**00-00-2006 to 00-00-2006**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A Loop-Free Algorithm Based on Predecessor Information	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**9**	

update messages, changes in the link-cost, link failures and link recoveries are processed one at a time in the order in which they occur.

Each node is represented by a unique identifier. Any link cost can vary in time but is always positive. The distance between two nodes in the network is measured as the sum of the link costs of the shortest path between the nodes.

A node failure is modeled as all the links incident on that node failing at the same time. A change in the operational status of a link or a node is assumed to be notified to its neighboring nodes within a finite time. These services are assumed to be reliable and are provided by the lower level protocols.

A *path* from node i to node j is a sequence of nodes where (i, n_1), (n_x, n_{x+1}), (n_r, j) are links in the path. A *simple path* from i to j is a sequence of nodes in which no node is visited more than once. The paths between any pair of nodes and their corresponding distances change over time in a dynamic network. At any point in time, node i is connected to node j if a path exists from i to j at that time. The network is said to be connected if every pair of operational nodes are connected.

The successor to destination j for any node is simply referred to as the successor of the node, and the same reference applies to other information maintained by a node. Similarly, updates, queries and responses refer to destination j, unless stated otherwise.

3 LPRA Description

Each node maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table is a matrix containing the distance and the predecessor entries for all destinations through all the neighboring nodes. The routing table is a column vector containing the minimum distance entries for each destination and its corresponding predecessor and *successor* (or next hop) information, which can be extracted from the distance table. The implicit path information from a node to any destination node can be extracted from the predecessor entries of the node's distance and routing tables. The link-cost table lists the cost of each link adjacent to the node; the cost of a failed link is considered to be infinity.

LPRA is specified in pseudocode form in Figure 1. Procedures *Init1* and *Init2* are used for initialization. Topology changes, handling of the feasibility condition (FC), path traversal in the distance table and updating of the routing tables are done by procedures *linkUp*, *linkDown*, *linkChange*, *FC*, *in_path* and *rt_update* respectively. Procedures *Update*, *Query* and *Reply* handles the updates, queries and replies respectively.

An update message consists of the source and the destination node identifiers, the path information and an update flag for one or more destinations. The update flag indicates whether the received message entry is an update (flag=0), query (flag=1) or a response (flag=2). The time at which an update message was sent and the number of hops the message has traversed so far can be recorded as well. The distance entry in a query is always set to ∞.

In LPRA, a node can be in one of the two states for each destination, *passive* or *active*. A node is passive if it has a *feasible successor*, or has determined that no such node exists and is active if it is searching for a feasible successor. A feasible successor for node i with respect to destination j is a neighbor node that satisfies the *feasibility condition* (FC). When node i is passive, it reports D^i_j in all its updates and replies. However, while the node is active, it sends an infinite distance in its replies and queries. Feasibility condition (FC) is defined as follows:

If at time t node i needs to update its current successor, it can choose as its new successor $s^i_j(t)$ from any node $n \in N_i(t)$ such that i is not present in the implicit path to j reported by neighbor n, $D^i_{jn} + d_{in}(t) = D_{min} = Min\{D^i_{jx} + d_{ix}(t) | x \in N_i(t)\}$ and $D^i_{jn} < FD^i_j(t)$. If no such neighbor exists and $D^i_j = \infty$, node i must keep its current successor. If $D_{min} = \infty$ then $s^i_j(t) = null$.

The successor graph for destination $j \in G$, denoted by $S_j(G)$, is a directed graph where nodes are the same nodes of G and where directed links are determined by the successor entries in the nodal routing tables. Loop-freedom is guaranteed at all times in G if $S_j(G)$ is always a directed acyclic graph.

When a node i processes an update from its neighbor k for destination j, the distance and predecessor entries in the distance table are updated (Step 1). Also, node i determines whether the path to destination j through any of its other neighbors $\{b \in N_i | b \neq k\}$ includes node k. If the path implied by the predecessor information reported by node b to destination j includes node k, then the distance entry of that path is updated as $D^i_{jb} = D^i_{kb} + D^k_j$ and the predecessor entry is updated as $p^i_{jb} = p^k_j$ (Step 2). Thus, a node can determine whether or not the update received from k affects its other distance and routing table entries.

When node is in the passive state and detects a cost change in link (i, k) or receives an update or query from neighbor k, it first updates its link-cost table with the new value of the link d_{ik}, its distance table with $D^i_{jk} = D^k_j + d_{ik}$ and $p^i_{jk} = p^i_k$. After that, it determines whether or not the node has a feasible successor, *i.e*, a neighbor node that satisfies FC.

If a feasible successor is found, node i updates its distance and predecessor entries of its distance table by traversing through all its other neighbors (Step 2) and then updates the routing table entries accordingly. Node i sends update messages to its neighbors if the routing table entries have changed. Furthermore, node i sets FD^i_j equal to the smaller of the updated value of D^i_j and the present value of FD^i_j.

If node i does not find a feasible successor, then FD^i_j is set to ∞. In addition, node i keeps its current path information and updates the distance entry in the distance table (Steps 1 and 2). If $D^i_j(t) = \infty$, then $s^i_j(t)$=null. Node i also sets the reply status flag ($r^i_{jk} = 1$) for all $k \in N_i$ and sends a query to all its neighbors. Node i is then said to be *active*, and cannot change its path information until it receives all the replies to its query. If the input event was a query from its neighbor after which i became active, node i responds to its neighbor k with the distance of its reply set to ∞. A node need not query its neighbor node k due to which node i became active. Instead, a node can send a

reply with infinite distance. If node k becomes passive after receiving node i's reply and changes its path information, then this change will be notified to node i by an update message.

When node i is active and receives a reply from node k, it updates its distance table and resets the reply flag ($r^i_{jk} = 0$).

Node i becomes passive at time t when $r^i_{jk}(t) = 0$ for every $k \in N_i$. At that time, node i can be certain that all its neighbors have processed its query reporting an infinite distance and node i is therefore free to choose any neighbor that provides the shortest distance, if there is any; or node i has found a feasible successor through one of its neighbors $k \in N_i$. If such a neighbor is found, node i updates the routing table as the minimum in distance-table row for destination j and also updates $FD^i_j = D^i_j$.

A node need not have to wait indefinitely for replies from its neighbors because a node replies to all its queries regardless of its state. Thus, there is no possibility of deadlocks due to the interneighbor coordination mechanism.

If node i is passive and has already set $D^i_j = \infty$ and receives an input that implies an infinite distance to j, then node i simply updates D^i_{jk} and d_{ik} and sends a reply to node k with an infinite distance if the input event is a query from node k. This ensures that updates messages will stop in G when a destination becomes unreachable.

Node i initializes itself in passive state with an infinite distance for all its known neighbors and with a zero distance to itself. After its initialization, node i sends updates containing the distance to itself to all its neighbors.

When node i establishes a link with a neighbor k, it updates its link-costs table and assumes that node k has reported infinite distances to all destinations and has replied to any query for which node i is active; furthermore, if node k is a previously unknown destination, node i initializes the path information of node k and sends an update to the new neighbor k for each destination for which it has a finite distance. When node i is passive and detects that link (i, k) has failed, it sets $d_{ik} = \infty$, $D^i_{jk} = \infty$ and p^i_{jk}=NIL; after that, node i carries out the same steps used for the reception of a link-cost change message in passive state. When node i is active and loses connectivity with a neighbor k, it resets the reply flag and resets the path information *i.e,* assumes that the neighbor k sent a reply reporting an infinite distance.

The addition or failure of a node is handled by its neighbors as all the links connecting to that node coming up or going down at the same time.

In contrast to LPRA, which makes a node i check the consistency of predecessor information reported by *all* its neighboring nodes each time an update is processed, Humblet's and other earlier path finding algorithms [1, 10, 8] check the consistency of the predecessor information only for the neighbor associated with the input event. In LDR [5], the path information is sent to the neighbors by an explicit label. Unlike that, in LPRA, path information is extracted by the predecessor information; Therefore, fixed size update entries can be used, rather than variable size entries that can contain the complete path in some cases.

As an example of LPRA's operation and its loop-freedom property, consider the five-node network depicted in Figure 2. In this network, links and nodes have the same processing or propagation delays; Q represents the queries,

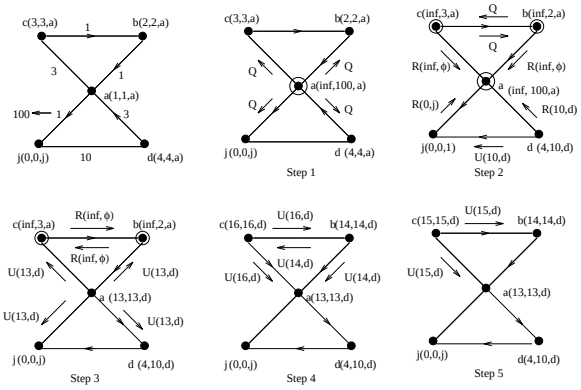

Figure 2: Example of the Algorithm's Operation

R replies and U indicates updates. The working of the algorithm is explained when the cost of the link (a, j) changes. The arrowhead from node x to node y indicates that node y is the successor of node x towards destination j (*i.e,* $s^x_j = y$). The label in parenthesis assigned to node x indicates the feasible distance from x to j (FD^x_j), current distance (D^x_j), and predecessor of the path from x to j (p^x_j). Steps 1 through 5 of Figure 2 depicts the behavior of LPRA. Update and reply messages are followed by the value of rd^x_j and rp^x_j in parentheses. Nodes in the active state are indicated with a circle around them. FD^i_j is always decreasing as long as node i is in the active state.

When node a detects the change in the cost of the link (a, j), it determines that it does not have a feasible successor as none of its neighbors have a distance smaller than FD^a_j = 1. Accordingly, node a becomes active by setting the reply status flag r^a_{j*} for all its neighbors for destination j and sends a query to all its neighbors (Step 1).

Nodes b and c also recognize that they do not have a feasible successor. This is achieved in a single step as the node traces through all its neighbors on receipt of an input event (Step 2 of the pseudocode). Node b (c) becomes active and sends query to c (b) and reply to a. On the other hand, node d is able to find a path to j and replies with the cost of the alternate path to j to node a's query and updates its distance to j maintaining the same feasible distance.

When node a receives replies from all its neighbors, it becomes passive again, and replies to the queries of nodes b and c with its feasible distance. Having found their feasible successor, nodes b and c update their path information accordingly. All nodes exchange update messages informing the new path information with their neighbors (Step 4) and the final stable topology is shown in Step 5.

4 Correctness of LPRA

Time T is defined such that links and nodes can go up and down and the cost of links can change within time interval [0,T). At time T, the nodes adjacent to each link have updated their link cost and status information, and there are no more changes after time T. An upstream node

```
Procedure Init1
when node i initializes itself do begin
    set a link-state table with the current costs of the links
    set N = i; N_i = {x | d_ix < ∞};
    for all (x ∈ N_i) do begin
        call Procedure Init2(x);
        for all (j ∈ N) do begin
            set D^i_jk = ∞; p^i_jk = null;
            set r^i_jk = 0
        end
    end
    set s^i_i = i, p^i_i = i, DT^i_i = FDT^i_i = 0;
    for each (n ∈ N_i)
    do add update U(i) to end of LIST_i(n);
    call Procedure Send
end
Procedure Init2
begin
    set N = N ∪ x;
    set p^i_j = i, s^i_j = null;
    set D^i_x = FD^i_x = ∞
end
Procedure Link_Down(i, k)
(5) when link (i, k) fails do begin
    delete column for neighbor k in D_i;
    d_ik = ∞; r^i_jk = 0;
    if (D^i_j ≠ ∞)
    then go to Step (3);
end
Procedure Link_Up(i, k, d_ik)
(6) when link (i, k) comes up do begin
    insert column k in D_i; update link-cost table;
    set r^i_jk = 0;
    Execute Procedure Update(k)
    Send the entire routing table to k;
end
Procedure Link_Change(i, k)
(7) when d_ik changes value do begin
    update link-cost table; update distance table;
    execute Procedure Update(k)
end
Procedure Send
begin
    for each (n ∈ N_i) do begin
        if (LIST_i(n) is not empty)
        then send message M_i[LIST_i(n)] to n
        Empty LIST_i(n)
    end
end
Procedure Message
when node i receives a message begin
    for each entry in the message
    do begin
        Check the type of the message
        0: [Entry is an update message]
            Call Procedure Update(k)
        1: [Entry is an query]
            Call Procedure Query(k)
        2: [Entry is an reply]
            Call Procedure Reply(k)
    end
end

Procedure In_Path(i, k, j, b)
begin
    set h ← p^i_j,b;
    if (h = i)
    then return(false)
    else begin
        if (h = k)
        then return(true)
        else In_Path(i, k, h, b);
    end
end
```

```
Procedure FC
(10) begin
        compute V_1 = {n | n ∈ N_i, D^i_jn < FD^i_j};
        compute D_min = Min{D^i_jx + d_ix | x ∈ N_i};
        compute V_2 = {n | n ∈ V_1, D^i_jn + d_in = D_min};
        if (V_2 = φ)
        then return (false);
        else return (true);
end


Procedure RT_Update
begin
    initialize all destinations to be unmarked
    for any unmarked destination j do begin
        if there is no finite distance in row j
        then mark j as undetermined
        else begin
            TV ← φ;
            for each node having minimum distance from i to j
                if (k = successor)
                then maintain the same path and successor
                else begin
                    c ← p^i_jb, TV ← TV ∪ c;
                    repeat c ← p^i_cb, TV ← TV ∪ c;
                        until D^i_cb is not minimum of row c or p^i_cb = i or p^i_cb
                        is marked
                    if ((p^i_cb is marked as undetermined) or
                    (D^i_cb is not minimum of row c))
                    then mark each node in TV as undetermined;
                    else begin
                        mark each node in TV as determined
                        D^i_j ← D^i_jb; p^i_j ← p^i_jb; s^i_j ← b;
                    end
                end
            end
        end
    end
    copy the routing vector to V^i if the distance or the
    predecessor has changed
end


Procedure Reply(k)
(9) begin
        set D^i_jk = rd^k_j + d_ki; p^i_jk = rp^k_j;
        set r^i_jk = 0;
        if (r^i_jk = 0, ∀n ∈ N_i)
        then begin
            if (FC is satisfied)
            then begin
                if ((D^i_j ≠ ∞) ∨ (D^i_jk ≠ ∞))
                then begin
                    set FD^i_j = D_min
                    Call Procedure RT_Update;
                    Add Updates to LIST_i(x), ∀x ∈ N_i;
                end
            end
            else begin
                set D^i_j = ∞; p^i_j = null;
            end
        end
        else begin
            if (k = s^i_j)
            then begin
                set D^i_j = rd^k_j + d_ik; p^i_j = rp^k_j;
            end
        end
end
```

```
Procedure Update(k)
when vector V^k,i is received on link (i, k)
do begin
    (0) begin
        RTEMP^i ← φ; DTEMP^i,b ← φ ∀ neighbors b
    end
    (1) for each triplet (j, D^k_j, p^k_j) in V^k,i, j ≠ i
    do begin
        D^i_jk ← D^k_j + d_ik; p^i_jk ← p^k_j;
        (2) for all neighbors b
            if k is in the path from i to j in the distance
            table through neighbor b
                D^i_jb ← D^i_kb + D^k_j; p^i_jb ← p^k_j
    end
    (3) if (r^i_jx = 0, ∀x ∈ N_i)
    then if (FC is satisfied) begin
            if ((D^i_j ≠ ∞) ∨ (D^i_jk ≠ ∞))
            then begin
                set FD^i_j = D_min;
                Call Procedure RT_Update
                Add updates to LIST_i(n), ∀n ∈ N_i
            end
    end
    else begin
        FD^i_j = ∞; r^i_jk = 1, ∀k ∈ N_i;
        if (k = null)
        D^i_j = ∞; p^i_j = null;
        else D^i_j = D^k_j + d_ik; p^i_j = p^k_j;
        Add queries to LIST_i(k), ∀k ∈ N_i;
    end
    (4) else begin
            if (k is the successor)
            then begin
                set D^i_j = D^k_j + d_ik;
                set p^i_j = p^k_j;
            end
    end
end
Procedure Query(k)
(8) begin
        set D^i_jk = rd^k_j + d_ik; p^i_jk = rp^k_j;
        if (j = i)
        then add reply with D^i_i to LIST_i(k)
        else begin
            if (r^i_jx = 0, ∀x ∈ N_i)
            then if ((D^i_j = ∞) ∧ (D^i_jk = ∞))
                    then add reply with D^i_j to LIST_i(k);
                    else begin
                        if (FC is satisfied)
                        then add reply with D^i_j to LIST_i(k))
                        else begin
                            set D^i_j = D^i_js^i_j + d_is^i_j; p^i_j = p^i_js^i_j;
                            for each n ∈ N_i − k do begin
                                set r^i_jn = 1; FD^i_j = ∞;
                                Add query to LIST_i(n)
                            end
                            set r^i_jk = 0;
                            Add reply with ∞ to LIST_i(k)
                        end
                    end
            end
            else begin
                if (k = s^i_j)
                then begin
                    set D^i_j = rd^k_j + d_ik; p^i_j = rp^k_j;
                end
                Add reply with ∞ to LIST_i(k);
            end
        end
end
```

Figure 1: LPRA Specification

with respect to node i is a node which is in the path from source i to destination j.

Proposition 1 If a loop is formed in the successor graph $S_j(G)$ for the first time at time t, then some node $i \in C_j(t)$ must choose an upstream node as its successor at time t.

By definition, the successor graph $S_j(G)$ is a directed acyclic graph before the loop is formed at time t. If a loop has to be formed at time t, there must be at least one node $k \in S_j(G)$ that changes its successor because the successor information can be changed only when an update occurs or when the node detects a change in a link cost or status. This implies that an upstream node will be chosen by some node $x \in C_j(t)$. □

Lemma 1 LPRA is free of deadlocks.

Proof: A node i can be in one of the two states — active or passive at any given time. A node receives updates, queries or replies at any given time.

When the node is in the active state, if a query is received from a neighbor, the node replies to these queries with an infinite distance since the node is still in the process of computing its path information at that time (active state). The node updates its distance table entries when either an update or reply message is received in active state.

When the node in passive state receives a message from its neighbor, it computes the feasible distance and updates its distance and routing tables accordingly and replies to the neighbors' queries with its feasible distance to the destination. If a feasible distance is not found, the node forwards the query to is neighbors and sends a reply with an infinite distance entry to the originator of the query.

Since a node always replies to the neighbor's queries, deadlock cannot occur. □

Lemma 2 The change in the cost or status of a link will be reflected in the distance and the routing tables of a node adjacent to the link within a finite time after time T.

Proof: The change in the link cost can be due to the link coming up, the link going down, or the link changing the link cost.

When a link comes up, a new column entry is added to the distance table and the new link-cost is assigned to the corresponding entry in the distance table (Step (6)). Procedure RT_Update is called, which eventually updates the routing table entry.

When a link goes down, the column entry is deleted and the distance entries in the distance table will be set to ∞ (Step (5)). Procedure RT_Update again updates the routing table entries accordingly.

When the link cost changes, the distance entry in the distance table is updated to reflect the new link cost (Step (1) and Step (2)). These changes are updated in the routing table again by Procedure RT_Update.

By assumption, any changes that occur in the time interval $[0, T)$ are updated by time T; this implies that the link cost changes are reflected in the distance and routing tables of a neighbor node within a finite time T. □

Lemma 3 Within a given finite time interval $[0, t)$, the number of shortest distance values to a given destination in a routing table is finite.

Proof: There is a finite number of links in the graph with a finite number of link cost changes within a given time interval $[0, t)$. Therefore, the cardinality of the set is finite. Also, the number of destinations in the graph are finite. This implies, there are finite number of entries stored in the routing table of each node.

The value of the shortest distance from any node in G to a given destination at time $t^{'}$ is equal to the cost of the link or the sum of the distance from the successor node to the destination and the link cost to the successor node or infinity if a node does not have a successor. This implies that there are finite number of values from a node to its destination in the time interval $[0, t)$. □

Theorem 1 *A finite time after time T, the routing table structures at all nodes form the final shortest path.*

Proof: The proof consists of showing that the old distance and predecessor information present in any one node's routing and distance tables are updated, and that the shortest-path trees are eventually computed.

A change in the cost of a link results in a routing table update by the nodes associated with that link (procedure RT_Update) as required. When a node has to select a new path, all the minimum distance entries in the distance table are found. These minimum distance entries can be obtained by finding the minimum of the distance table entries for a given destination. Also, the loop freedom of the implicit path can be ensured by the predecessor information present in the node's routing and distance tables (procedure in_path). This implies that K, the number of hops in the path is finite. If the current successor is the same as the successor for any of these minimum distance entries, then the current path information is retained. The resulting path is the shortest path in the final graph; otherwise, the new path information is computed and the routing table is updated accordingly.

It follows from the design of LPRA that, if node i does not have a path to node j, the distance in the routing table is set to ∞. Because the number of hops K is finite, and $H(i, d)$, the maximum number of links in the path from i whose distance to any node is less than or equal to d in the final topology is finite, all paths have a final shortest path.

Let the initial time be $T(0) = T$. Let $T(K)$ be the time at which all messages that are in transit at time $T(K-1), K \geq 0$, have arrived at their destination and have been processed. The rest of proof is done by induction on K. At time $K = 0$, because there are no messages to be transmitted, the theorem holds true. Assume that the theorem is true for some $K = M > 0$.

A path of $M+1$ links is the concatenation of an adjacent link and a path with M links. From the definition of $T(K)$, by time $T(M + 1)$, the routing trees computed at time $T(M)$ have all been communicated to the neighbors of the nodes which computed such trees. By Lemma 2, these link cost changes are updated in the node's distance and routing tables. This proves the theorem.□

Lemma 4 No node in the network G can be active after a finite time t for any destination.

Proof: Consider a network G with a stable topology. We need to consider three cases: a node receives a query from

its neighbor, a node receives an update message from its neighbor, and a node observes a link-cost change.

When a node receives a query from its neighbor, regardless of the state of the node, the node responds to the query immediately either by a feasible distance (if it is in passive state) or by sending an infinite distance (if it is in active state). Therefore, the node cannot remain active for an infinite time when it receives a query from a neighbor.

When an update is received by a node i, and if i has a feasible successor, it remains in the *passive state*, updates its distance table entries within a finite time (step 1 in the algorithm) and proceeds accordingly. If the node is unable to find a feasible successor, it becomes active by sending queries to its neighbors. Since a node responds to all the queries within a finite time, the update process will also be done in a finite time.

When node i observes a link-cost change, the distance table at i is updated and the routing table is updated as defined by the feasibility condition. The updated entries at i are sent to neighbors N_i as updated messages. Since update messages are processed in finite time, link-cost changes are also processed in finite time. A node failure is treated as all the links incident on the failed node going down at the same time and the link cost changes can be treated as a link going down and coming up with a new link cost. Therefore, this will also be processed within a finite time.

Therefore, a node in G can be active only for a finite time for any destination j. □

Lemma 5 If at time t all nodes in the graph G are reachable from one another then, a finite time after t, no new update messages are being transmitted or processed by any node, and the entries corresponding to each destination j in all the distance and routing tables are correct.

Proof: Given that the graph G is finite, there is a finite number of entries in a node's distance and routing tables. From Lemma 4, we also have that a node cannot be active for an indefinitely long period of time. Assume that at time t, all nodes in G are reachable.

After time t, let node i receive an update message due to the change in the link-cost. This results in sending update messages to all the neighbors $k \in N_i$. From Lemma 4, the update message will be processed in finite time after receiving it. Also, from Theorem 1, we have correct entries in the node's distance and routing tables when the algorithm converges. This proves Lemma 5. □

Lemma 6 Assume that at time t at least one node $j \in G$ is inaccessible to a connected subset of nodes $A_j \subset N$. Then, no node $i \in A_j$ can terminate for node j with $D_j^i < \infty$.

Proof: This lemma is proved by contradiction.

Assume that node $i \in A_j$ does terminate LPRA with a finite distance at time $t_s > t$; this implies that s_j^i is constant and D_j^i is constant and finite since time t_s. Let $s_j^i(t_s) = k$; then because all link costs are constant after time t, $D_j^i(t_s) = D_j^k(t_s) + d_{ik}(t)$. It is obvious that, if any node $i \in A_j$ terminates at time t_s, its successor $s_j^i(t_s)$ must terminate before that time. Following the same argument, and traversing the path i to j, it must be true that each node x in such a path must have a constant distance $D_j^x(t_s)$. Thus, because $|A_j|$ and $D_j^i(t_s)$ are finite and LPRA is free of deadlocks and loops [12], a node n must be reached for which $D_j^n(t_s) = l_j^n(t)$ and $s_j^n(t_s) = j$; this is impossible because j is not adjacent to any node in A_j at time $t \leq t_s$ and every node knows its neighbors. Therefore, the lemma is true. □

Lemma 7 If there is at least one node $j \in G$ that is inaccessible to a subset of nodes in G at time t then, a finite time after t, no new update messages with an entry for node j will be transmitted or processed by nodes, and the entries corresponding to node j in all topology and routing tables are correct.

Proof: By contradiction.

After time t, G must consist of one or more connected components and a set of zero or more isolated nodes. Because an isolated node sets all its routing-table entries to infinity after detecting that it has no neighbors, the proof needs to consider only connected components.

Assume that, after time t, node j is unreachable from a set of nodes that constitutes a connected component of the graph A_j. The proof needs to show that every node $x \in A_j$ sets $D_j^x = \infty$ a finite time after time t and, therefore, cannot change that entry to a finite value until a new topological change occurs.

If node $n \in A_j$, LPRA can not terminate with a finite distance (Lemma 6). If A_j is connected, it follows from the way in which distances and successors are updated that either all the nodes in A_j terminate with an infinite distance, or each of them keeps sending updates reporting changes in its distance or update label indefinitely.

Assume that the nodes in A_j do not terminate, and consider the case in which node $i \in A_j$ never sets $D_j^i = \infty$ after a finite time $t_i \geq t$. If this is the case, given that every node in A_j has a finite number of neighbors and that no topology changes occur after time t, node i must choose at least one neighbor node that satisfies FC, say node s, as its successor an infinite number of times after time t_i. If this is true for node i, the same must apply for node s and for all the nodes in at least one path from node i to node j. However, every node knows its neighbors, which implies that after a finite time $t_f \geq t$ all paths in A_j must lead to nodes who have set their distances equal to ∞. Therefore, every node in A_j must send queries (reporting an infinite distance) an infinite number of successions after a finite time greater than or equal to time t. From the assumption, no node in A_j terminates, and each node has a finite number of neighbors, it follows that node i must choose some neighbor node, say s, that satisfies FC as its successor and report the path information to all its neighbors $x \in N_i$ after receiving all the replies to its queries. This must occur an infinite number of times which, however, is impossible.

Node $i \in A_j$ has a path to the destination j only if i has $D_j^i < \infty$. This is not possible after a finite time larger than or equal to time t, as all the nodes know their neighbors and it is assumed that there are no topological changes after time t. Accordingly, after a finite time $t_f \geq t$, whenever node i updates its routing table with a finite distance value, it must update its path information also. Therefore, for node i to be able to set $D_j^i < \infty$ an infinite number of successions after

time t, there must exist an infinite number of paths (even if they are incomplete) from node i to node j in $S_j(\bar{A}_j)$ after time t, which is impossible because $\bar{A}_j$ has a finite size.

From the above, it follows that every node $x \in \bar{A}_j$ sets $D_j^x = \infty$ a finite time after time t, and the lemma is true. □

Theorem 2 *A finite time after t, no new update messages are being transmitted or processed by nodes in G, and all entries in distance and routing tables are correct.*

Proof: By contradiction.

Assume that the transmission of update messages reporting topological changes never ceases or terminates with incorrect values in the routing tables. This implies that there must be at least one row (call it j) of the routing tables for which either an infinite number of update messages are generated or incorrect information is obtained. After time t, either all nodes are mutually reachable or at least one is inaccessible from a subset of nodes in the graph. As Lemmas 5 and 7 are true for any destination j, the theorem is true. □

5 Performance of LPRA

The overhead required to converge to correct routing table entries in the worst case are computed assuming that the algorithm behaves synchronously so that every node in the network executes a step of the algorithm simultaneously at fixed points in time. At each step, the node receives and processes all the inputs originated during the preceding step and if required, sends update messages to the neighboring nodes at the same step. The first step occurs when at least one node detects a topological change and issues update messages to its neighbors. During the last step, at least one node receives and processes messages from its neighbors and after which the node stops transmitting any update messages till a new topological change has taken place. The number of steps taken for this process is termed as *time complexity* (TC) and the number of messages required to accomplish this is the *communication complexity* (CC).

DBF has a worst-case time complexity of $O(|N|)$ and worst-case communication complexity of $O(|N^2|)$, where, N is the number of nodes in the network G. In contrast, LPRA can be shown to have TC=O(x) and CC=O(x) [12] where, x is the number of nodes affected by a single topology change.

To obtain an insight into the average performance of LPRA, the algorithm was analyzed by simulation using the topologies of typical networks. Simulations were performed using *Drama* [13] along with a network simulation library. The performance was compared with a diffusing update algorithm DUAL [4] and an ideal link-state algorithm (ILS). The simulation uses link weights of equal cost, and zero link transmission delays. During each simulation step, a node processes input events received during the previous step one at a time, and generates messages as needed for each input event it processes. To obtain the average figures, the simulation makes each link (node) in the network fail, and counts the steps and messages needed for each algorithm to recover. It then makes the same link (node) recover and repeat the process. The average is then taken over all link (node) failures and recoveries. The results of this simulation for Los-Nettos are shown in Table 1. The table shows the total number of events (updates and link-status changes processed by nodes) (EC), the total number of update messages transmitted (PC), the total number of steps needed for the algorithms to converge (DUR), and the total number of operations performed by all the nodes in the network (OC).

Table 1: Simulation Results for Los-Nettos

Para-	LPRA		DUAL		ILS	
meter	mean	sdev	mean	sdev	mean	sdev
Link Failure						
EC	88.6	43.3	49.9	18.6	29.0	5.8
PC	33.7	16.6	32.6	11.8	27.0	5.8
DUR	5.4	1.7	6.7	1.3	4.2	0.88
OC	77.6	25.6	69.9	18.6	724.1	27.3
Link Recovery						
EC	83.7	7.0	45.7	7.4	33.3	1.9
PC	15.1	3.8	17.0	7.2	31.9	1.9
DUR	2.8	0.6	3.7	0.9	3.86	0.5
OC	97.1	18.8	65.7	7.5	944.0	45.8
Node Failure						
EC	123.8	22.96	67.6	19.7	31.8	9.6
PC	47.3	8.9	45.7	3.3	26.7	7.19
DUR	5.9	1.1	7.0	1.0	4.09	0.51
OC	115.8	27.35	113.6	40.1	702.9	204.3
Node Recovery						
EC	175.4	98.3	86.7	34.3	56.2	13.4
PC	26.9	10.4	39.0	11.2	51.1	10.8
DUR	3.6	0.83	4.7	0.5	4.4	0.5
OC	213.1	99.2	132.7	56.13	1698.8	478.4

The details of simulation analysis appear in [12]. However, it is worth noting that, as expected, LPRA and DUAL have better overall average performance than ILS after the recovery of a single node or a link. The simulation results also indicate that, insofar as overhead traffic is concerned, the average performance of LPRA is comparable to DUAL and ILS. LPRA converges faster than DUAL in all cases; in particular, LPRA is more responsive in the case of node failures, which is a concern in DUAL's performance [4]. CPU utilization on ILS is two orders of magnitude larger than in LPRA and DUAL. On the other hand, LPRA converges in almost the same number of steps as ILS after link and node failures. Accordingly, LPRA constitutes a more scalable solution for routing in large internets than ILS and even DUAL.

6 Conclusions

In this paper, we have presented a routing algorithm (LPRA) that eliminates the formation of temporary routing loops without the need for internodal synchronization

spanning multiple hops or the specification of complete path information. LPRA works on the notion of using information about the second to last hop of shortest paths to ensure termination, and an efficient interneighbor coordination mechanism to eliminate temporary loops. A detailed proof of the correctness and loop-freedom of LPRA is presented and its complexity is analyzed. The worst-case complexity of LPRA for single recovery/failure is $O(x)$, x being the number of nodes affected by this recovery/failure. LPRA is compared with DUAL and an ideal link state algorithm (ILS) by simulation. The simulation results show that LPRA converges faster than DUAL for single-resource changes and the number of messages exchanged is comparable to DUAL.

References

[1] C. Cheng, R. Reley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect", *ACM Computer Communications Review*, Vol.19, No.4, 1989, pp.224–236.

[2] L.R. Ford and D.R. Fulkerson, *Flow in Networks*, Princeton University Press, Princeton, New Jersey, 1962.

[3] J.M. Jaffe and F.M. Moss, "A Responsive Routing Algorithm for Computer Networks", *IEEE Trans. Comm.*, Vol.30, July 1982, pp.1758–1762.

[4] J. J. Garcia-Luna-Aceves, "Loop-free Routing using Diffusing Computations", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 1, Feb, 1993, pp.130–141.

[5] J. J. Garcia-Luna-Aceves, "Distributed Routing with Labeled Distances", *IEEE Infocom*, Vol.2, May 1992, pp.633–643.

[6] J. J. Garcia-Luna-Aceves, "LIBRA: A Distributed Routing Algorithm for Large Internets", Proceedings of *IEEE Globecom*, Vol.3, Dec 1992, pp.1465–1471.

[7] J. Hagouel, "Issues in Routing for Large and Dynamic Networks," IBM Research Report RC 9942 (No. 44055) Communications, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, April 1983.

[8] P.A. Humblet, "Another Adaptive Shortest-Path Algorithm", *IEEE Trans. Comm.*, Vol.39, No.6, June 1991, pp.995–1003.

[9] P.M. Merlin and A. Segall, "A Failsafe Distributed Routing Algorithm", *IEEE Trans. Comm.*, Vol.27, Sept. 1979, pp.1280–1288.

[10] B. Rajagopalan and M. Faiman, "A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems," *Internetworking: Research and Experience*, Vol.2, No.1, March 1991, pp. 51-69.

[11] M.S. Sloman and X. Andriopoulos, "A Routing Algorithm for Interconnected Local Area Networks", *Computer Networks and ISDN Systems*, 1985, pp.109–130.

[12] Shree Murthy, "Design and Analysis of Distributed Routing Algorithms", *Master's Thesis*, University of California, Santa Cruz, June 1994.

[13] W. T. Zaumen, "Simulations in Drama", *Network Information System Center, SRI International*, Menlo Park, California, January 1991.